GRADUATE THESIS FIELD OF STUDY
09.04.01 – «COMPUTER SCIENCE»

ACADEMIC PROGRAM TITLE:
«DATA SCIENCE»

Development of source code summarization methods aimed at verification of accomplishment of requirements towards to software products

Performed by: Viacheslav Karpov
Thesis supervisor: Vladimir Ivanov

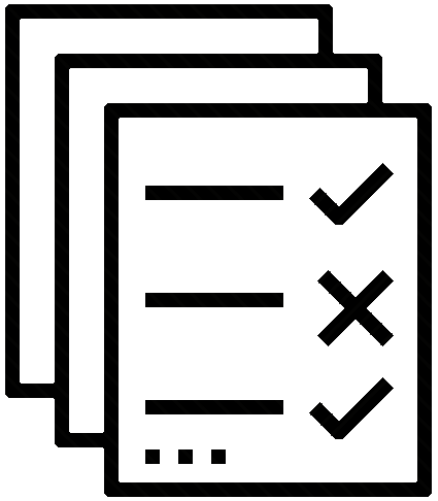
Summary

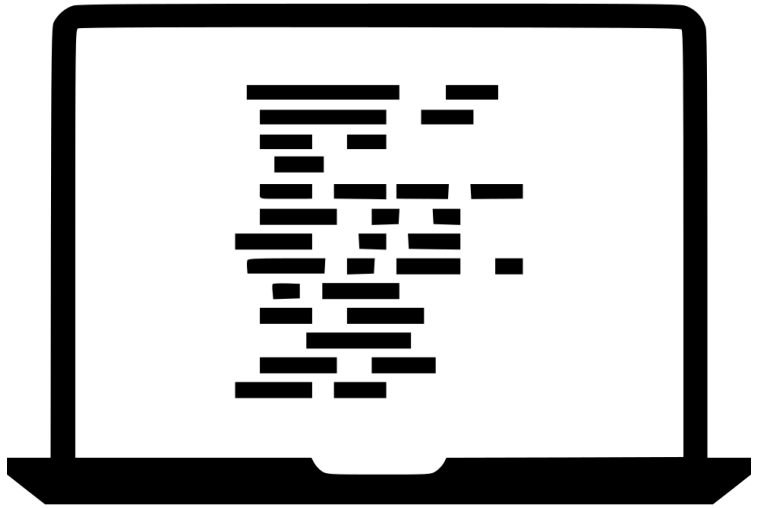
- Problem specification
- Related work
  - Source code summarization
  - Source code retrieval
- Research questions
- Proposed solution
- Evaluation and discussion
- Demo
- Conclusion

Motivation example

Problems:

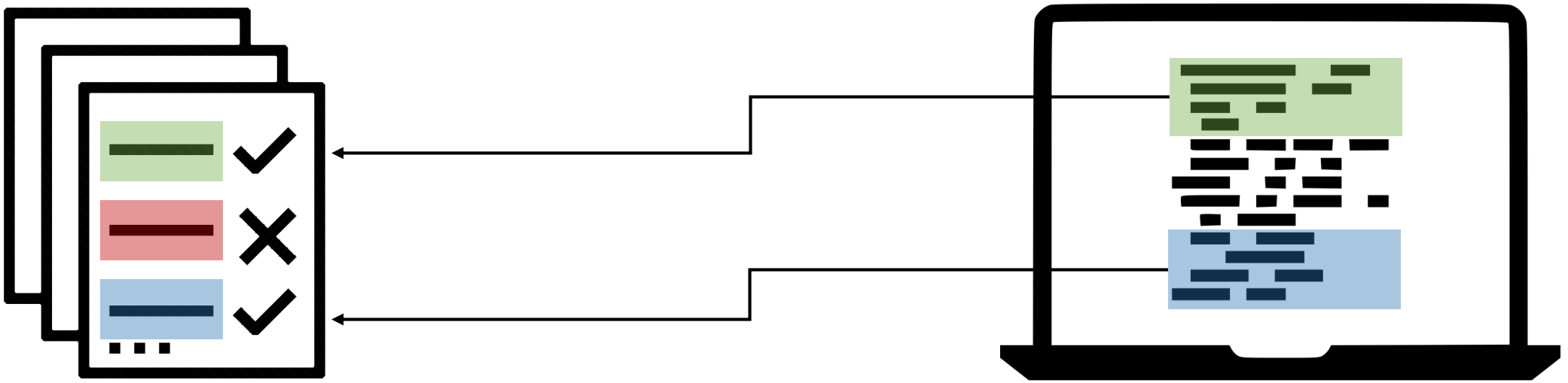
- How to verify an accomplishment of requirements towards to software product?
- How to retrieve the product parts with desired functionality?





Problem specification

- Given a functional requirement R and an entire product implementation P retrieve the software artifacts $A_1, \ldots, A_r$ expected to accomplish the requirement.
- Based on the proximity between A_1 to R tell if the requirement is satisfied.



Problem importance

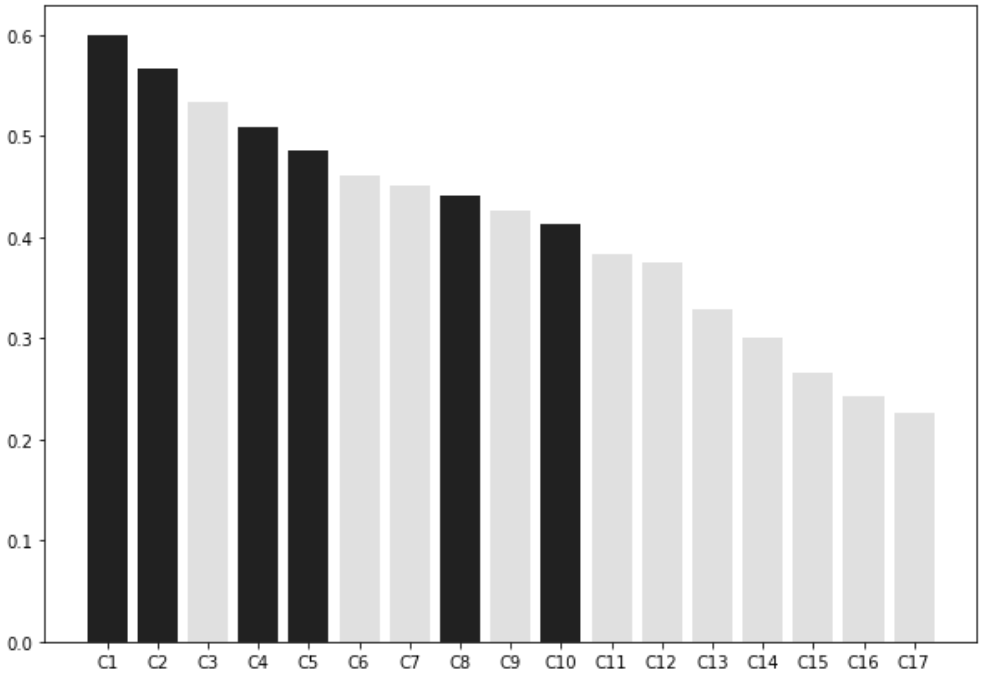


Figure 1. Frequencies of product failures according to survey of J.V. Balsera [1], where black bars correspond to requirement related failures

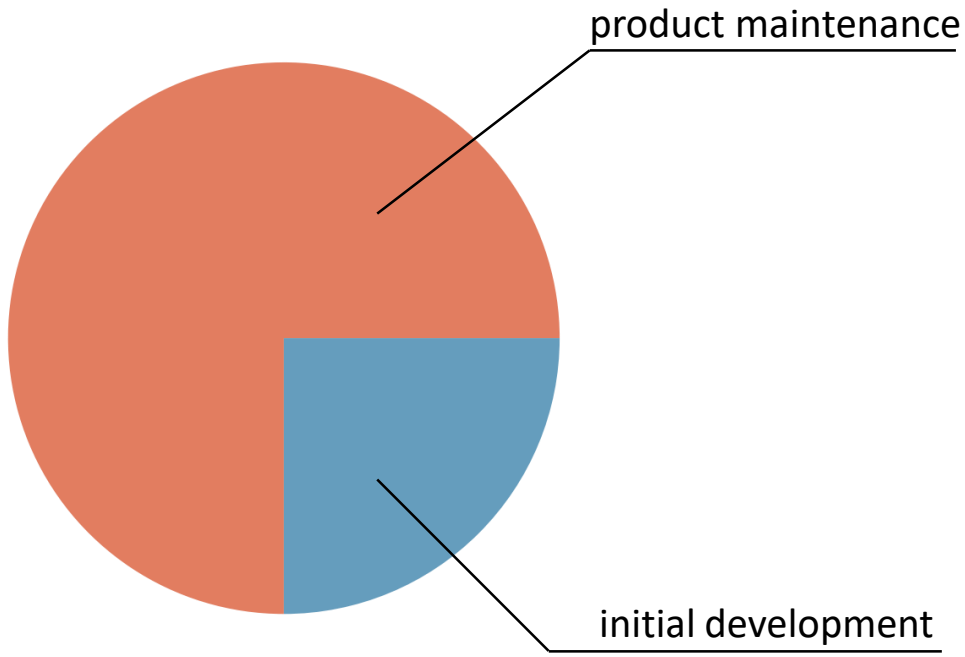


Figure 2. Ratio of costs of initial development and product maintenance according to B. Boehm [2]

Related work

- Source code summarization
  - Extractive summarization:
    - Regular expressions [E. Reiter and R. Dale, 2000]
    - Vector Space Model (VSM) [Haiduc et al., 2010]
    - Predefined templates [McBurney and McMillan, 2014]
  - Abstractive summarization:
    - **Latent Semantic Analysis (LSA)** and hPAM [Eddy et al., 2013]
    - **CNN** [Allamanis et al., 2016]
    - **RNN encoder-decoder** [Iyer et al., 2016]
- Source code retrieval
    - Neural Code Search (NCS) model [Sachdev et al., 2018]
    - **Unified Embedding (UNIF) model** [Cambronero et al., 2019]

Research questions

- **RQ 1:** How precisely can we refer the correspondence between software requirements written in natural language and related software artifacts?

- **RQ 2:** Which representation level of software artifacts is the most appropriate for determining the compliance between software artifacts and requirement specifications?

Proposed solution
Model structure

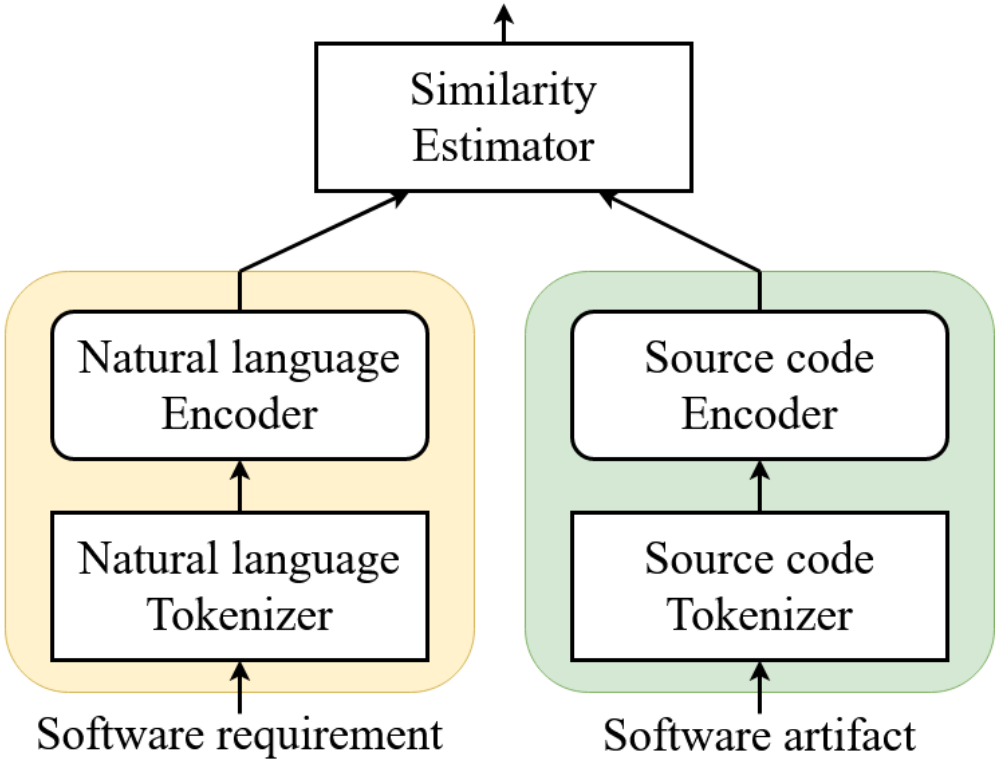
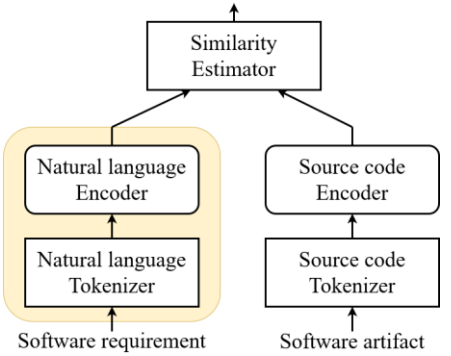


Figure 3. Simplified computational graph of proposed model

- Siamese Artificial Neural Network structure
- Expects an input of arbitrary many software requirements and artifacts
- Learns joint embeddings of inputs
- Estimates proximity in terms of distance between vector representations

Natural language branch
BERT encoder



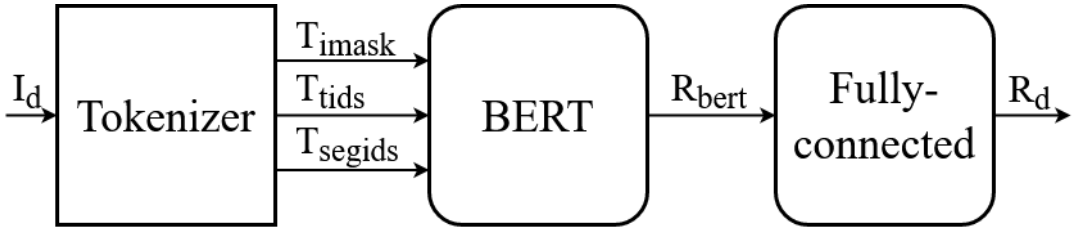
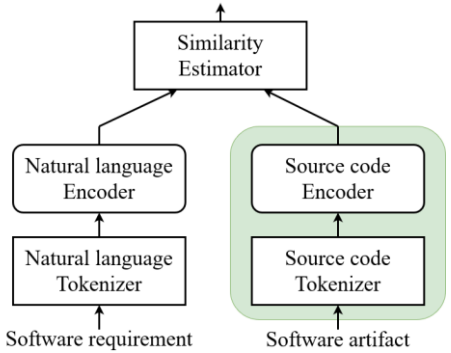


Figure 4. Natural Language Embedding block layout

- 110M parameters
- High computational complexity
- State-of-the-art representation model [11]
- Originally pre-trained on Wikipedia and Common Crawl corpora
- Expected to adapt the learned embeddings to our domain during fine-tuning

Source code branch
N-gram CNN encoder



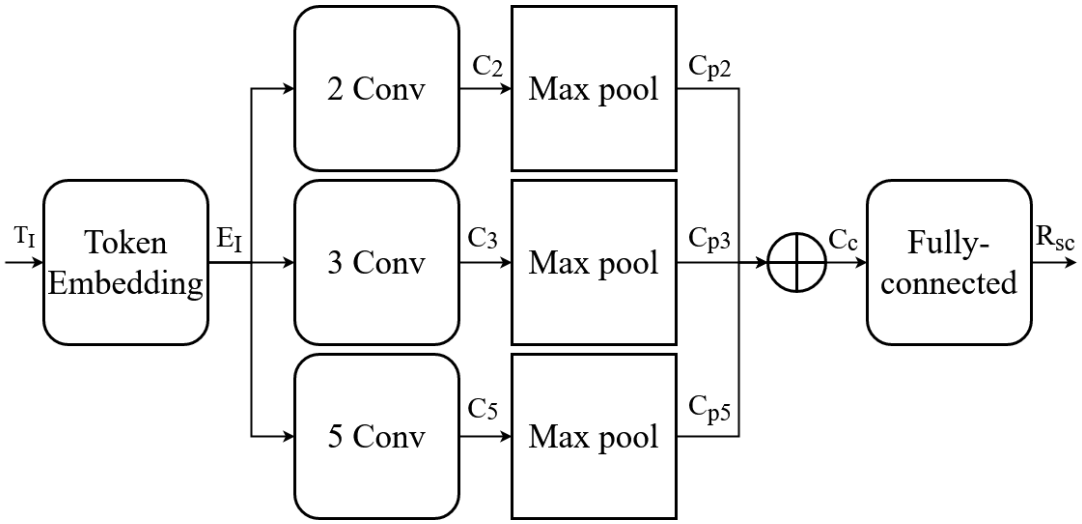
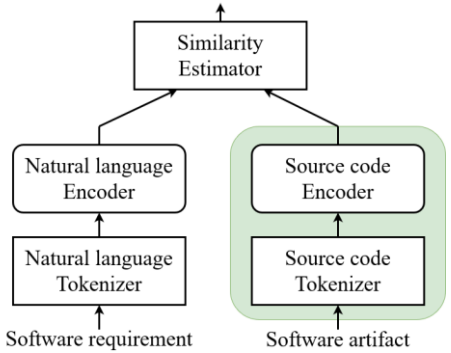


Figure 5. N-gram CNN encoder layout

- 2M parameters
- Bag-of-contexts representations
- Captures local features across several consecutive word
- Carries assumption, that long-term connections in source code are non-significant

Source code branch
Augmented API calls encoder



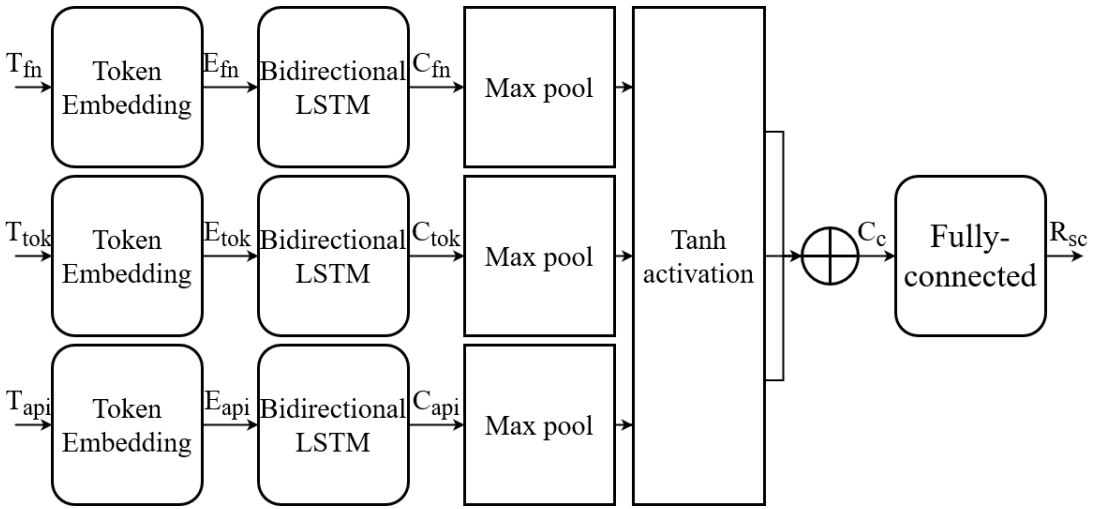
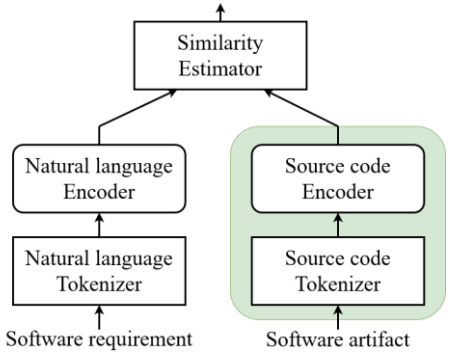


Figure 6. Augmented API calls encoder layout

- 7.6M parameters
- Representations on an extracted sequence of API calls
- Requires language specific tools to extract API call sequences
- Augmented with function name and body tokens representations
- Limited parallelization potential due to recurrent connections

Source code branch
Self-attention encoder



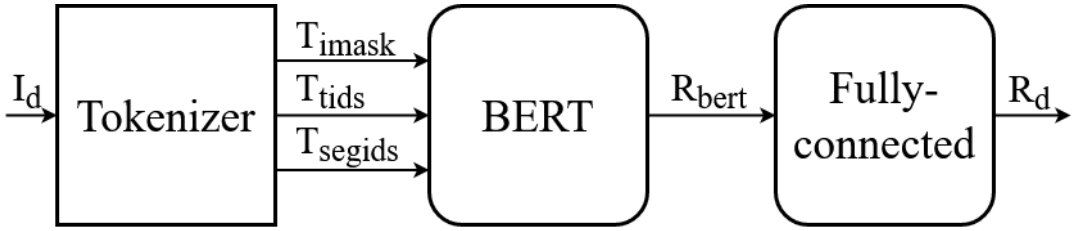
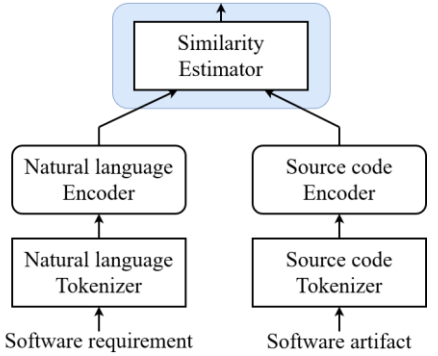


Figure 7. BERT based self-attention encoder layout

- 11.5M parameters
- Carries the highest computational complexity of source code embedding models
- Entire function level representations
- Truncated version of BERT [Devlin et al., 2018]
- Learnt from scratch

Similarity estimator block
Cosine similarity



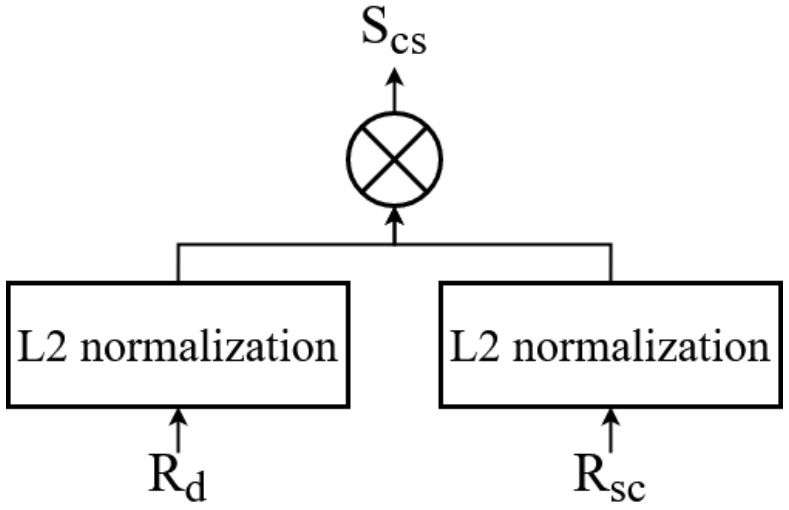


Figure 8. Cosine similarity estimator block layout

- No trainable parameters
- Scale-invariant but shift-sensitive
- Preserves the captured context of representations
- Used as the loss function
- Enables unsupervised learning with negative sampling

Training details

Dataset	Altered CodeSearchNet Collection [Husain et al., 2019]: pairs of Java methods with related docstring descriptions • 454,127 training pairs • 15,302 validation pairs • 26,909 test pairs
Assumptions	One functional requirement – One method An accomplishment can be estimated in terms of proximity
Training approach	Unsupervised learning with random sampling of negative pairs
Loss function	Clipped cosine similarity
Stopping criteria	No quality progress on the validation dataset for 5 iterations

Evaluation results

Evaluation metrics:

- MRR – mean multiplicative inverse of rank of the true relevant element across retrievals
- R@k – fraction of retrievals where the true relevant element appears among *k* first elements

Model	Trainable parameters	MRR	R@1	R@5	R@10
Tf-idf LSA	2.1M	0.32	0.23	0.46	0.59
Neural BOW	1.4M	0.41	0.31	0.54	0.67
N-gram CNN	2M	0.43	0.28	0.62	0.78
API-encoder	7.6M	**0.55**	**0.42**	**0.74**	**0.87**
Self-attention	11.5M	0.41	0.25	0.59	0.76

Table 1. Overall performance of baseline and applied models

Demo

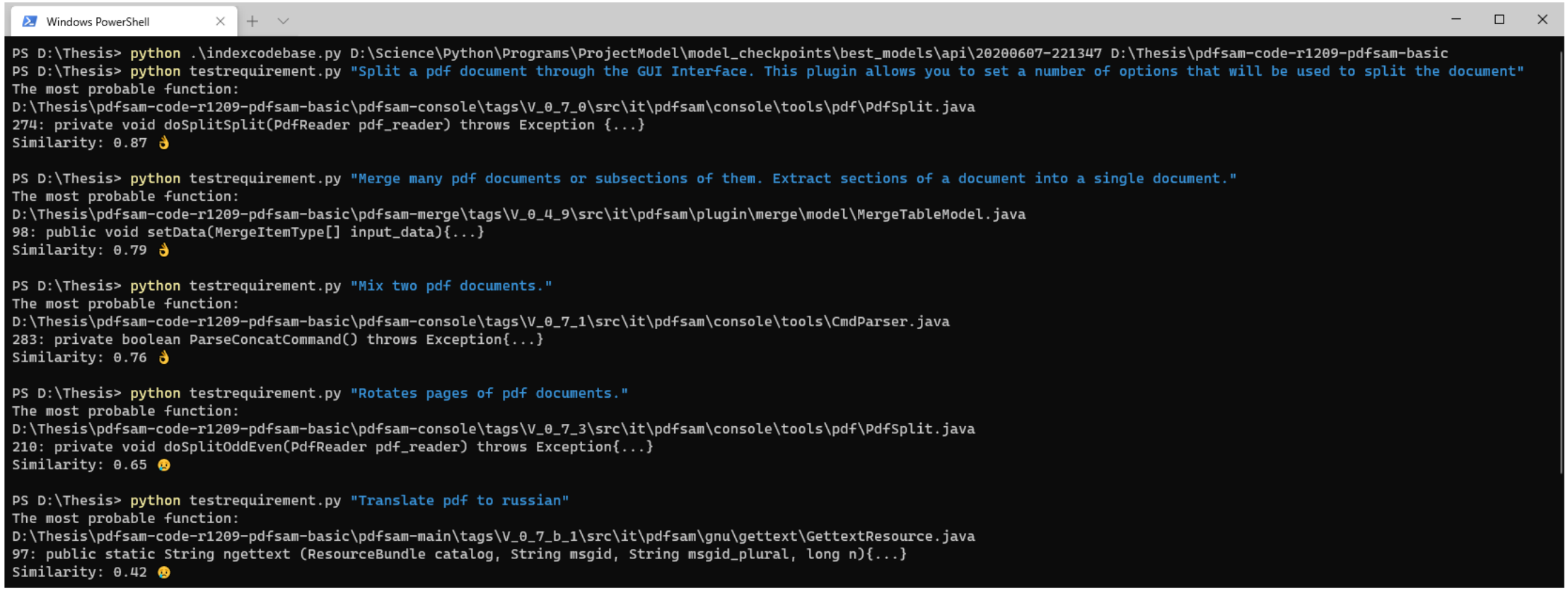


Figure 9. Test the requirements of “PDF Split and Merge” open source product

Conclusion

- Joint latent representations can be used to estimate proximity between natural language descriptions and source code of software artifacts
- The idea can be exploited to verify an accomplishment of software requirements
- In some cases a functional requirement can not be fulfilled by one function. Instead, an entire class can be dedicated to execute one requirement
- In future work we would like to adopt the model for class level artifacts

Link to GitHub repo with Source Code, Demo and Presentation

Bibliography

[1] Rodríguez Montequín, V., Villanueva Balsera, J., Cousillas Fernández, S. M., & Ortega Fernández, F. (2018). Exploring project complexity through project failure factors: Analysis of cluster patterns using self-organizing maps. Complexity, 2018.

[2] Barry Boehm, Chris Abts, and Sunita Chulani. "Software development cost estimation approaches—A survey." *Annals of software engineering* 10.1-4 (2000): 177-205.

[3] E. Reiter and R. Dale, "Building natural language generation systems", Cambridge university press, 2000.

[4] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context", Proceedings of the 22nd International Conference on Program Comprehension. – ACM, pp. 279–290, 2014.

[5] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of auto-mated text summarization techniques for summarizing source code", 17th Working Conference on Reverse Engineering. – IEEE, pp. 35–44, 2010.

[6] B. P. Eddy, J. A. Robinson, N. A. Kraft, and J. C. Carver, "Evaluating source code summarization techniques: Replication and expansion",21stInternational Conference on Program Comprehension (ICPC), 2013.

[7] M. Allamanis, H. Peng, and C. Sutton, "A convolutional attention net-work for extreme summarization of source code", International Conference on Machine Learning, pp. 2091–2100, 2016.

[8] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model", Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1:Long Papers), pp. 2073–2083, 2016.

[9] S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, and S. Chandra, "Retrieval on source code: A neural code search", Proceedings of the 2nd ACM SIG-PLAN International Workshop on Machine Learning and Programming Languages, pp. 31–41, 2018.

[10] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "When deep learning met code search", Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 964–974, 2019.

[11] J. Devlin, M.-W. Chang, K. Lee и K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", arXiv preprint arXiv:1810.04805, 2018.